Multi-scenario high-level synthesis for dynamic delay variation and its evaluation on FPGA platforms

Koki Igawa^{a)}, Masao Yanagisawa, and Nozomu Togawa^{b)}

Department of Computer Science and Communications Engineering,

Waseda University, Tokyo 169–8555, Japan

a) koki.igawa@togawa.cs.waseda.ac.jp

b) togawa@togawa.cs.waseda.ac.jp

Abstract: Multi-scenario high-level synthesis for distributed register/controller architecture has been proposed targeting static delay variation. In this paper, we extend it and propose a floorplan-driven high-level synthesis algorithm which can be applied to dynamic delay variation by effectively using an error prediction technique, where pre-error registers are introduced to local registers in every circuit block. Experimental results show that the proposed algorithm using two and three *scenarios* on an FPGA chip reduces the average number of required control steps by 17.6% and 25.5% on average compared to worst-case high-level synthesis at the expense of increasing lookup-tables and flip-flops. Moreover, we implement a multi-scenario elliptic-wave-filter (EWF) circuit with three scenarios synthesized by our proposed algorithm onto an FPGA chip and run it under the environment with varying supply voltages which causes dynamic delay variation. The FPGA implementation experiments also demonstrate that the EWF circuit effectively runs on the real FPGA chip. As far as we know, this is the world-first experiment where a multi-scenario circuit runs under real dynamic delay variation environment.

Keywords: high-level synthesis, dynamic delay variation, adaptive behavior, distributed register/controller architecture

Classification: Integrated circuits

References

- [1] M. Wirthofer: *Variation-Aware Adaptive Voltage Scaling for Digital CMOS Circuits*, Springer Series in Advanced Microelectronics (2013).
- [2] M. Zhao, *et al.*: “Joint profit and process variation aware high level synthesis with speed binning,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **23** (2015) 1640 (DOI: [10.1109/TVLSI.2014.2349493](https://doi.org/10.1109/TVLSI.2014.2349493)).
- [3] K. Kawamura, *et al.*: “A floorplan-aware high-level synthesis technique with delay-variation tolerance,” *IEEE International Conference on Electron Devices and Solid-State Circuits* (2015) 122 (DOI: [10.1109/EDSSC.2015.7285065](https://doi.org/10.1109/EDSSC.2015.7285065)).
- [4] J. Cong, *et al.*: “A variation-tolerant scheduler for better than worst-case behavioral synthesis,” *Proc. CODES+ISSS 2009* (2009) 221 (DOI: [10.1145/](https://doi.org/10.1145/)

- 1629435.1629467).
- [5] A. Del Barrio, *et al.*: “A distributed clustered architecture to tackle delay variations in datapath synthesis,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **35** (2016) 419 (DOI: [10.1109/TCAD.2015.2474362](https://doi.org/10.1109/TCAD.2015.2474362)).
 - [6] K. Igawa, *et al.*: “A multi-scenario high-level synthesis algorithm for variation-tolerant floorplan-driven design,” *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **E99-A** (2016) 1278 (DOI: [10.1587/transfun.E99.A.1278](https://doi.org/10.1587/transfun.E99.A.1278)).
 - [7] S. Abe, *et al.*: “Energy-efficient high-level synthesis for HDR architectures,” *IP SJ Trans. Syst. LSI Design Methodol.* **5** (2012) 106 (DOI: [10.2197/ipsjtsldm.5.106](https://doi.org/10.2197/ipsjtsldm.5.106)).
 - [8] T. Sato and Y. Kunitake: “A simple flip-flop circuit for typical-case designs for DFM,” *Proc. International Symposium on Quality Electronic Design* (2007) 539 (DOI: [10.1109/ISQED.2007.23](https://doi.org/10.1109/ISQED.2007.23)).
 - [9] HLSynth92 benchmark. <http://ftp.ics.uci.edu/pub/hlsynth/HLSynth92/>.
 - [10] SAKURA. <http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html>.

1 Introduction

High-level synthesis is required to tackle with complex and large-scale IC designs. At the same time, the problems to consider at design time are increasing due to decrease in device feature sizes. Among them, delay variation and interconnection delays are the major problems because they affect timing design significantly. The worst-case design is a technique to deal with these problems where design margins are added assuming worst-case conditions. It realizes high yield but may degrade the performance of synthesized circuits. In order to solve this difficulty, we have to focus on interconnection delays and delay variation even in high-level synthesis.

Delay variation can be classified into static delay variation and dynamic delay variation: Static delay variation is mainly caused by process variation; Dynamic delay variation is mainly caused by supply voltage variation, temperature variation and aging effect. In particular, supply voltage variation includes not only voltage noise but intentional voltage change such as multiple voltage modes and adaptive voltage scaling [1].

Several static-delay-variation aware high-level synthesis algorithms are proposed [2, 3]. But most of them assume only static delay variation due to process variation and cannot be applied to dynamic delay variation. Dynamic-delay-variation aware high-level synthesis algorithms are also proposed [4, 5] which realize dynamic recovery from timing errors by stalling. However, they do not consider interconnection delays, which can cause the timing problem in their recovering phases.

To incorporate delay variation together with interconnection delays, floor-plan-driven multi-scenario high-level synthesis has been proposed [6]. It targets distributed register/controller architecture [7] that can deal with interconnection delays by abstracting RTL modules in high-level synthesis and integrate it with floorplanning easily. Moreover, a circuit synthesized by the multi-scenario high-level synthesis algorithm has multiple behaviors (called *scenarios*) depending on its delay condition. Switching them depending on its manufactured condition, the

chip can operate adaptively to delay variation. However, it targets only static delay variation. If we apply [6] to dynamic delay variation, we have to switch its behavior before timing error occurs. We require the strategy to apply a timing error prediction technique to [6].

In this paper, we propose a floorplan-driven high-level synthesis algorithm considering dynamic delay variation. We focus on pre-error flip-flops (FFs) [1, 8] as a timing-error prediction technique and incorporate them into the multi-scenario high-level synthesis algorithm proposed in [6].

Experimental results show that the proposed algorithm using two and three scenarios on an FPGA chip reduces the average number of required control steps by 17.6% and 25.5% on average compared to worst-case high-level synthesis at the expense of increasing lookup-tables (LUTs) and FFs. Moreover, we implement a multi-scenario elliptic-wave-filter (EWF) circuit with three scenarios synthesized by our proposed algorithm onto an FPGA chip and run it under the environment with varying supply voltages which causes dynamic delay variation. The FPGA implementation experiments also demonstrate that the EWF circuit effectively runs on the real FPGA chip. As far as we know, this is the world-first experiment where a multi-scenario circuit runs under real dynamic delay variation environment.

2 Floorplan-driven high-level synthesis for dynamic delay variation

We use distributed register/controller (DRC) architecture [7] as in Fig. 1 for realizing floorplan-driven high-level synthesis efficiently. DRC architecture distributes local registers (LR) and local controller (LC) to each functional unit (FU) and then interconnection delays among them can be effectively estimated. A *circuit block (CB)* is composed of FUs, LRs and LC. In each CB, FUs, LRs and LC are packed and their detailed placement and routing inside CB is not given. CB is small enough so that the interconnection delays inside CB is ignored. For simplicity, we assume that each CB includes just a single FU in this paper. Interconnection delays between CBs are explicitly estimated based on CB floorplanning and hence we can consider interconnection delays efficiently in high-level synthesis while reducing the complexity of placement.

2.1 Multi-scenario high-level synthesis

Scenario is defined as a pair of a control-data flow graph (CDFG) and its operating condition. By preparing multiple scenarios depending on delay variation and synthesizing them onto a single chip, it can behave adaptively to manufactured conditions or operating environment. Here we assume a CDFG in every scenario is identical, i.e., a synthesized circuit has only a single application but its execution condition varies depending on a scenario (See Figs. 4(a)–4(c)).

A multi-scenario high-level synthesis problem for DRC architecture can be defined as follows [6]:

Definition 1. *Given a set of FUs, clock period constraint, and a set of scenarios, each of which is composed of CDFG and its associated module delays, find a scheduling and binding result for all the scenarios onto a single chip satisfying*

Fig. 1. DRC architecture.

interconnection delays between circuit blocks. The primary objective is to minimize the number of required control steps for each scenario. The secondary objective is to minimize the chip area.

The synthesis flow proposed in [6] is shown in Fig. 2. Multi-scenario scheduling/FU-binding finds a scheduling/FU-binding result to minimize the number of control steps for each scenario. Multi-scenario register binding finds a register binding result to minimize the number of required registers for each scenario. In these two processes, the *commonization process* is also performed for reducing the area overhead of synthesizing multiple scenarios. Multi-scenario controller synthesis synthesizes a controller for each CB which can control and switch between scenarios. In floorplanning, each CB is floorplanned so that timing constraints between CBs are satisfied.

2.2 Applying multi-scenario high-level synthesis to dynamic delay variation

Assume that we have two scenarios, one of which is for typical-case delay scenario and the other one is for worst-case delay scenario. When a circuit synthesized by the multi-scenario high-level synthesis algorithm runs under the typical-case delay scenario and it predicts timing errors somewhere, it switches the current scenario to the other one. In this case, we must switch from the current scenario to the other one *before* timing error occurs and hence a *timing error prediction technique* must be introduced into it.

In this paper, we focus on pre-error FFs [1, 8] as a timing error prediction technique. Assume that a signal path from a source register to a destination register. In this case, the destination register based on a pre-error FF can output a timing error prediction signal when it predicts a timing error in the near future. As one of pre-error FFs, a canary FF [8] can be used which is composed of two FFs, a delay-line, and a comparator as in Fig. 3(a). A register composed of pre-error FFs is called a *pre-error register*. We replace some of LRs in DRC architecture with pre-error registers and thus it becomes possible to predict the timing errors while operating a particular scenario. Fig. 3(b) shows the operating environment for the circuit based on DRC architecture generated from multi-scenario high-level synthesis utilizing pre-error registers (which is called a *multi-scenario circuit*).

When transferring the operation results from a source FU to its sink LR, a timing error prediction signal is output from the sink LR before the timing error actually occurs. The timing error prediction signal is sent to an on-chip scenario

Fig. 2. Multi-scenario high-level synthesis flow for DRC architecture.

Fig. 3. Multi-scenario circuit equipped with pre-error registers.

controller, which sends back a scenario control signal to the multi-scenario circuit. After that, the multi-scenario circuit switches its scenario to a new one and hence the timing error does not actually occur.

2.3 Requirement for multi-scenario register binding

Let us assume that a multi-scenario circuit has the three scenarios as depicted in Fig. 4. In this example, the circuit has the scenarios $s1$, $s2$, and $s3$ and we can complete their operations in 4, 6, and 8 steps, respectively.

Fig. 4(d) shows the conventional multi-scenario register binding result. Let us consider the case where the timing error prediction signal (*pre-error signal* in short) is observed in the data transfer to the register $r1$ when operating the scenario $s1$. In this case, we cannot know which one of the signal line, (A) from the multiplier to the register $r1$ or (B) from the adder to the register $r1$, causes the pre-error signal. If the signal line (A) causes the pre-error signal, we should switch from the scenario $s1$ to the scenario $s2$ to enlarge the execution time of the multiplier. On the other hand, if the signal line (B) causes the pre-error signal, we should switch from the scenario $s1$ to the scenario $s3$ to enlarge the execution time of the adder. We have to decide which one of (A) and (B) causes the pre-error signal.

Fig. 4. A motivational example for modifying multi-scenario register binding.

The simple solution to this problem is that, the scenario controller receives the pre-error signal as well as the select signals to the selector associated with r_1 and decides the next appropriate scenario based on all of these signals. However, this solution has the problems as below:

Problem 1: When the number of multiplexers and/or the number of their selector bits become larger, a scenario controller should receive more signals from a multi-scenario circuit and thus it should have too many input ports. Moreover, the scenario controller can become more complex and its critical path delay may cause a serious problem.

Problem 2: If interconnection delay variation cannot be negligible, it is possible that pre-error signals and select signals do not arrive at a scenario controller in a single clock cycle. In this case, the scenario controller may fail to decide its next appropriate scenario to switch. The number of interconnections between a multi-scenario circuit and a scenario controller must be as small as possible.

In order to solve the above problems effectively, we propose modified multi-scenario register binding as shown Fig. 4(e), where each register to be observed is bound exclusive to every FU. When we observe the pre-error signal from the register r_1 in this figure, we can know that it is caused by the data transfer from the

multiplier to r_1 . In this case, we can switch from the scenario s_1 to the scenario s_2 to enlarge the execution time of the multiplier.

In the modified register binding, since the scenario controller just receives the signals from registers to be observed, the scenario controller and registers are connected by the minimum number of signals and thus the problems above are effectively solved.

2.4 Multi-scenario register binding exclusive to FU

Let us consider an FU f_i and a scenario x_h . Then we replace the $I_{f_i}^{x_h}$ sink registers of f_i with pre-error registers in the scenario x_h . We assume that $I_{f_i}^{x_h}$ is given beforehand for every FU f_i .

The proposed multi-scenario register binding algorithm exclusive to a source FU is summarized as follows:

- Step 1:** Perform multi-scenario register binding based on [6] to generate a conventional register allocation and binding result for each scenario.
- Step 2:** Let $slack_{x_h}(f_i, r_j)$ be the slack time of the data transfer in the scenario x_h from FU f_i to LR r_j , which can be calculated based on [6]. Let $R_{x_h}(f_i)$ be a set of sink LRs from FU f_i in the scenario x_h .
- Step 3:** For every LR $r \in R_{x_h}(f_i)$, replace r with a pre-error register in the ascending order of $slack_{x_h}(f_i, r)$ in each scenario x_h . We replace totally $I_{f_i}^{x_h}$ registers with pre-error registers here in every scenario.
- Step 4:** Share the registers and pre-error registers as much as possible whenever register binding is exclusive to each FU.

3 Experimental results

We have implemented our algorithm in C++ and run on Windows 7 environment. After multi-scenario high-level synthesis, we use Xilinx ISE 14.6 for logic synthesis and layout synthesis. Our target FPGA device is Spartan-6 (XC6SLX75, Speed Grade: -2). At high-level synthesis, we set the clock period constraint to be 16.6 ns.

The supply voltages range from 0.7 V to 1.1 V. We set the *typical-case delay* to be the delay at 1.1 V and the *worst-case delay* to be the delay at 0.7 V. We use DCT [7], FIR [7], and EWF [9] as experimental applications. All the FUs used in these applications have bit width of 16 bits. We assume three scenarios and the FU delay conditions in each scenario are summarized in Table I (See Section 3.1.3 on how to obtain this condition). In this table, “*typ*” and “*wst*” show the typical-case and worst-case delays, respectively. In this experiment, for each FU f_i in the scenarios s_1 and s_3 , we set $I_{f_i}^{s_1} = 1$ and $I_{f_i}^{s_3} = 0$. For each f_i in the scenario s_2 , we set $I_{f_0}^{s_2} = 1$, $I_{f_1}^{s_2} = 1$ and $I_{f_2}^{s_2} = 0$. This parameter shows that the LR in the most critical path for each FU is replaced with the pre-error register and thus these paths are monitored when each application runs.

For simplicity, we assume the conditions below:

1. All necessary input data are ready before an application program is executed.
2. When we use Xilinx ISE, we set Keep Hierarchy constraint in every application so that the CB hierarchy is kept.

Table I. The FU delay conditions in each scenario.

App.	Scenario	Add1	Add2	Add3	Mul1	Mul2	Mul3
EWF	s1	<i>typ</i>	<i>typ</i>	-	<i>typ</i>	-	-
	s2	<i>typ</i>	<i>typ</i>	-	<i>wst</i>	-	-
	s3	<i>wst</i>	<i>wst</i>	-	<i>wst</i>	-	-
DCT	s1	<i>typ</i>	<i>typ</i>	<i>typ</i>	<i>typ</i>	<i>typ</i>	<i>typ</i>
	s2	<i>typ</i>	<i>typ</i>	<i>typ</i>	<i>wst</i>	<i>wst</i>	<i>wst</i>
	s3	<i>wst</i>	<i>wst</i>	<i>wst</i>	<i>wst</i>	<i>wst</i>	<i>wst</i>
FIR	s1	<i>typ</i>	<i>typ</i>	<i>typ</i>	<i>typ</i>	<i>typ</i>	<i>typ</i>
	s2	<i>typ</i>	<i>typ</i>	<i>typ</i>	<i>wst</i>	<i>wst</i>	<i>wst</i>
	s3	<i>wst</i>	<i>wst</i>	<i>wst</i>	<i>wst</i>	<i>wst</i>	<i>wst</i>

Table II. RTL module delay.

	Delay [ns]	
	Typical-case	Worst-case
Adder	0.993	5.958
Multiplexer	2.865	17.19
Register	0.599	3.594
2-to-1 Multiplexer	0.235	1.410

3.1 Experimental setup

3.1.1 FPGA platform

We assume SAKURA-G [10] as the experimental FPGA board, on which Spartan-6 chip is embedded. In SAKURA-G board, the power supply to the Spartan-6 chip is given through the voltage regulator whose output is adjusted by the variable resistance. By adjusting the value of the variable resistance, we can change the supply voltage to it dynamically.

3.1.2 Pre-error flip-flops

We use a canary FF [8] as a pre-error FF as shown in Fig. 3(a). The delay line of the canary FF is implemented by connecting four inverters in series. Large area overhead may become a problem by using canary FFs, but in this paper we focus not on the efficient structure of pre-error FFs but only on the function of pre-error FFs.

3.1.3 Modeling of delay variation

The typical-case delays and worst-case delays of RTL modules (adder, multiplier, register, and 2-to-1 multiplexer) used in our algorithm are summarized in Table II. We set the typical-case delay to be the delay obtained from timing report by Xilinx ISE.

However, the module delay at 0.7 V cannot be obtained by Xilinx ISE. We estimate the module delay at 0.7 V by measuring the oscillation cycle when varying supply voltages to a ring oscillator as shown in Fig. 5. According to Fig. 5 and noise margin, we set the worst-case delay to $(\text{typical-case delay}) \times 6$.

Fig. 5. Oscillation cycle vs supply voltage of the ring oscillator on Spartan-6.

Table III. High-level synthesis and logic synthesis results.

Application	FUs	Algorithm	HLS result				LS result		
			#CSs			Average #CSs	Minimum period [ns]	#LUTs	#FFs
			s1	s2	s3				
EWF	Add×2	Worst (s3)	-	-	32	32.0 (1.00)	11.295	738	442
	Mul×1	Ours (s1,s3)	16	-	32	26.7 (0.83)	13.755	1354	842
		Ours (s1,s2,s3)	16	21	32	23.0 (0.72)	13.755	1568	884
DCT	Add×3	Worst (s3)	-	-	22	22.0 (1.00)	11.973	1692	847
	Mul×3	Ours (s1,s3)	11	-	22	18.3 (0.83)	15.376	3688	2016
		Ours (s1,s2,s3)	11	14	22	15.7 (0.71)	15.407	4331	2197
FIR	Add×3	Worst (s3)	-	-	24	24.0 (1.00)	11.731	1168	921
	Mul×3	Ours (s1,s3)	12	-	23	19.3 (0.81)	15.892	3164	1672
		Ours (s1,s2,s3)	12	23	23	19.3 (0.81)	15.892	3169	1672

3.2 High-level and logic synthesis results

We first perform high-level synthesis (HLS) to obtain RTL circuits and, after that, we perform Xilinx ISE to obtain logic synthesis (LS) results. The results are shown in Table III.

“Worst (s3)” shows the results obtained by [7] on which the conventional worst-case (single-scenario) high-level synthesis is performed. “FUs” shows a set of FUs given as input. “Average #CSs” shows the average number of required control steps of all the synthesized scenarios assuming the probabilities of operating each scenario is equal. “Ours (s1,s3)” and “Ours (s1,s2,s3)” show the results obtained by the proposed multi-scenario high-level synthesis algorithms with two and three scenarios, respectively. The used pre-error registers in “Ours (s1,s3)” and “Ours (s1,s2,s3)” are summarized in Table IV.

Comparing to “Worst (s3)”, “Ours (s1,s3)” and “Ours (s1,s2,s3)” reduce “Average #CSs” by 17.6% and 25.5% on average. On the other hand, “Minimum clock period” is increased by 28.6% on average. The required LUTs (“#LUTs”) are increased by 124.1% and 146.6% on average. The required FFs (“#FF”) are increased by 103.4% and 113.6% on average.

Comparing “Ours (s1,s3)” and “Ours (s1,s2,s3)”, “Ours (s1,s2,s3)” has 11.1% more LUTs and 4.1% FFs on average. Increasing the number of scenarios does not cause much area overhead. The main reason to increase area overhead is introducing 16-bit canary FFs. A 16-bit pre-error register based on the canary FFs requires 64 more inverters and 16 more FFs than a normal 16-bit register, which

Table IV. The used pre-error register counts out of allocated register counts in every circuit block (CB).

App.	Algorithm	CB1	CB2	CB3	CB4	CB5	CB6
EWF	Ours ($s1, s3$)	3/12	3/11	3/11	-	-	-
	Ours ($s1, s2, s3$)	3/16	3/11	3/11	-	-	-
DCT	Ours ($s1, s3$)	6/15	6/15	6/14	4/12	4/12	4/12
	Ours ($s1, s2, s3$)	6/17	6/18	6/16	4/12	4/13	4/13
FIR	Ours ($s1, s3$)	6/8	6/9	6/9	0/16	0/16	1/16
	Ours ($s1, s2, s3$)	6/8	6/9	6/9	0/16	0/16	1/16

mainly causes to increase #FF and #LUT. How to implement pre-error registers with low-area overhead is one of the important future works.

3.3 EWF-Ours ($s1, s2, s3$) on real FPGA board under voltage variation

In this section, we implement the multi-scenario EWF circuit synthesized by our proposed high-level synthesis algorithm with the scenarios $s1$, $s2$, and $s3$ (EWF-Ours ($s1, s2, s3$)) on the Spartan-6 chip of SAKURA-G board and operate it under the voltage variation which causes dynamic delay variation. The scenarios used here are shown in Table I.

Clock frequency supplied from the on-board oscillator is 48 MHz and our FPGA environment is shown in Fig. 6. SAKURA-G board has two FPGA chips; one is our target FPGA (Spartan-6) chip on which EWF-Ours ($s1, s2, s3$) is implemented and the other is the I/O FPGA chip. A scenario controller is also implemented on the target FPGA chip. The behavior is shown in Fig. 6. We have the nine pre-error signals of $e0$ to $e8$ and the scenario switch occurs based on the condition C1 and the condition C2 in Fig. 6.

EWF-Ours ($s1, s2, s3$) iterates its process repeatedly. The initial scenario is set to be $s1$. The scenario in the i -th iteration is decided from the pre-error signals received in the $(i - 1)$ -th iteration. Then, the scenario controller determines the next scenario before the i -th iteration starts. The I/O FPGA chip has the interface circuits which we modify the reference design [10] to communication between the target FPGA chip and the host PC. Fig. 6 shows the floorplan results of three CBs of EWF-Ours ($s1, s2, s3$) visualized by Xilinx PlanAhead.

The floorplanned EWF-Ours ($s1, s2, s3$), scenario controller, and interface circuit are placed-and-routed together and then bitstreams are generated for target FPGA chip by Xilinx ISE.

We run the target FPGA chip as follows:

1. The host PC generates sixteen random 16-bit data for input to EWF and transmits them to the I/O FPGA chip. Simultaneously, they are input to the EWF software implemented on the host to generate correct outputs.
2. The I/O FPGA chip transfers input data to the target FPGA chip.
3. The scenario controller sets the appropriate scenario to EWF-Ours ($s1, s2, s3$) implemented on the target FPGA chip. EWF-Ours ($s1, s2, s3$) starts its iterations when all the input data are ready.

Fig. 6. Verification environment.

- After each iteration is done, the operation results are transferred to the host PC through the I/O FPGA chip.
- The verification program on the host PC compares the received results from EWF-Ours ($s1, s2, s3$) and the correct outputs calculated by the host PC. If they are not equal, we consider that the timing error occurs in the target FPGA chip.

We perform 1000 iterations and measure the timing error occurrence counts for every operating scenario per setting voltage.

3.3.1 Controlling scenarios manually

Firstly, we switch scenarios manually reducing the supply voltage from 1.1 V to 0.7 V gradually. The result is shown in Table V. When we give the supply voltage of 1.1 V, timing errors do not occur in the scenario $s1$. When we give the supply voltage of 0.8 V, timing errors do not occur in the scenario $s1$ but several timing error prediction signals are measured. Note that, even if some timing error prediction signals occur, they just do not switch scenarios but a scenario will be switched only if the conditions described in Fig. 6 are satisfied.

When we give the supply voltage of 0.75 V, timing errors occur 204 times out of 1000 iterations in the scenario $s1$. After switching the scenario from $s1$ to $s2$, no

Table V. The number of timing errors in various voltages out of 1000 iterations.

Supply voltage [V]		1.1	0.8	0.75		0.7	
Manually	Operating scenario	$s1$	$s1$	$s1$	$s2$	$s2$	$s3$
	# of errors (/1000)	0	0	204	0	487	0
Automatically	Operating scenario	$s1$	$s1$	$s2$		$s3$	
	# of errors (/1000)	0	0	0		0	

timing errors are observed. When we give the supply voltage of 0.7 V, timing errors occur 487 times in the scenario $s2$. After switching the scenario from $s2$ to $s3$, no timing errors are observed.

3.3.2 Controlling scenarios automatically

Secondly, we switch scenarios automatically using the scenario control circuit reducing the supply voltage from 1.1 V to 0.7 V gradually. At each supply voltage, the EWF application is iterated 1000 times. In this case, no timing errors are observed at the supply voltages of 1.1 V, 0.8 V, 0.75 V, and 0.7 V. Our multi-scenario circuit can behave adaptively to its dynamic delay condition caused by dynamic voltage variation by controlling scenarios appropriately.

All of these results demonstrate that our proposed algorithm successfully can apply to dynamic delay variations.

4 Conclusion

In this paper, we proposed a floorplan-driven multi-scenario high-level synthesis algorithm for dynamic delay variation. Our algorithm with two and three scenarios reduce the average number of required control steps by 17.6% and 25.5% on average compared to worst-case high-level synthesis. Moreover our approach on a real FPGA chip can operate adaptively to dynamic delay variation.

Acknowledgments

This research was supported in part by the MIC/SCOPE #141303001.